COLLABORATIVE ENGINEERING-DESIGN SUPPORT SYSTEM

Dong Ho Lee and D. Richard Decker

Electrical Engineering and Computer Science Department
19 Memorial Drive West
Lehigh University
Bethlehem, PA 18015

ABSTRACT

Designing engineering objects requires many engineers' knowledge from different domains. There needs to be cooperative work among engineering designers to complete a design. Revisions of a design are time consuming, especially if designers work at a distance and with different design description formats. In order to reduce the design cycle, there needs to be a sharable design description the engineering community, which can be electronically transportable. Design is a process of integrating that is not easy to define definitively. This paper presents Design Script which is a generic engineering design knowledge representation scheme that can be applied in any engineering domain. The Design Script is developed through encapsulation of common design activities and basic design components based on problem decomposition. It is implemented using CLIPS with a Windows NT graphical user interface. The physical relationships between engineering objects and their subparts can be constructed in a hierarchical manner. The same design process is repeatedly applied at each given level of hierarchy and recursively into lower levels of the hierarchy. Each class of the structure can be represented using the Design Script.

INTRODUCTION

Design is a fundamental purposeful human activity with a long history. Design can include creative artistic and engineering components. Knowledge-based design systems deal with factors that tend to be limited to the engineering aspects of design. Many researchers have developed knowledge-based engineering design systems. Many of these systems were developed for the specified design domain using their own unique problem solving method [2, 5, 9, 12, 21]. Some have tried to develop domain independent knowledge-based design systems. The DIDS (Domain Independent Design System) by Runkel [18] was developed as set of tools that can provide a configuration-design system from a library. DOMINIC [10] treats design as best-first search by focusing on the problem of iterative redesigning of a single object. GPRS (Generative Prototype Refinement Shell) was developed by Oxman [15], which used Design Prototype [8, 22] as a knowledge representation. In the real world, most engineering designs are so complex that a single individual cannot complete them without many other engineers' help. Cooperation between different engineering designers is not a simple process because each designer may have a different perspective for the same problem, and multiple revisions of a design are needed in order to finish a project. Designers may have different interpretations of the same design value or may want to access special programs to determine values for the design variables in which they are interested. In order to achieve the above goals, there needs to be a design knowledge

representation that can be shared between designers and that can be modified to the designers' needs. In addition, if a designer needs to execute a special program, the design system should provide a scheme to do so. This paper describes a Design Script as an abstract model of the design process that is based on hierarchical design structure and shows how to capture design knowledge and integrate data and tools into a knowledge based design system.

MODELS OF DESIGN PROCESSES

Dieter [7] describes the design process in his book as follow: "There is no single universally acclaimed sequence of steps that leads to a workable design." But it is possible to make the fundamental design process as simple as an iterative process of analysis, synthesis, and evaluation (Fig. 1). Analysis is required to understand the goals of the problem and to produce explicit statements of functions. The synthesis phase involves finding plausible solutions through the guidance of functions that are produced from the analysis phase. The evaluation process checks the validity of solutions relative to the goals. The evaluation phase can be divided into two different types of jobs. One is to compare the solution with existing data if the solution is composed of comparable data; and the other is to compare the solution values derived from the current design solution through simulation process executions with the given goals.

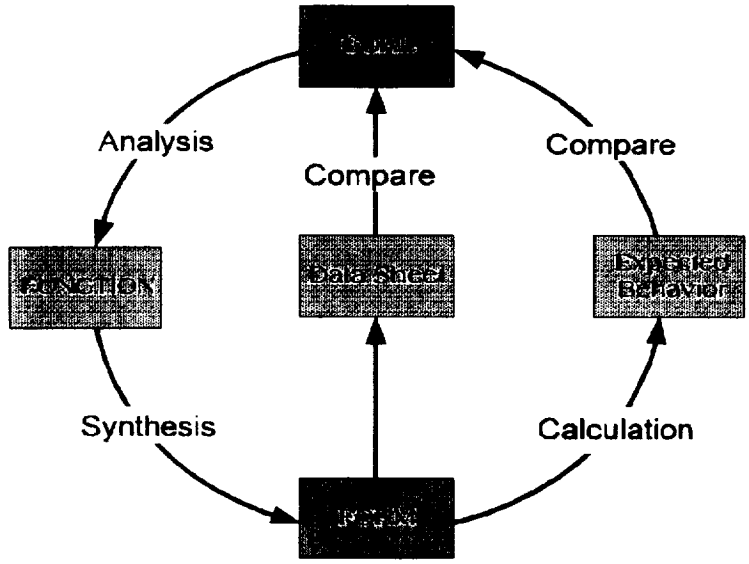

Fig. 1. Goal, Function, Form and Behavior Relationship

Problem decomposition is a well-known problem solving strategy in knowledge-based design systems. Once a complex design problem (a complete object) is decomposed into simple design problems (subparts), it is much easier to handle. The engineering design object is considered as being composed of sub-parts in a hierarchical structure. In other words, the problem of designing a complete object is comprised of designing number of subparts recursively until the designing process reaches the elemental subparts. The extent and complexity of a design problem can be different at different levels of hierarchy, but the identical design process can be applied at all hierarchical levels.

DESIGN SCRIPT

The DS (Design Script) is presented in this paper as an engineering design knowledge representation scheme. The DS is implemented in COOL (CLIPS Object Oriented Language) of NTCLIPS which is a Windows-NT version of CLIPS developed by the authors [20]. The common design process which is composed of analysis, synthesis, and evaluation phases can be abstracted by encapsulating design activities, such as goal decomposition and invoking methods. The abstracted model Design Script is placed at the top of the design knowledge for a domain. The major components of the DS are the goals, functions, and form of the part which is designed (Fig. 2). These abstractions include how the goals are decomposed and passed to lower level objects, how the functions of subparts are represented and used, and how explicit knowledge about reasoning and transforming process are formed and used. The instances of design sub-part can inherit these abstractions.

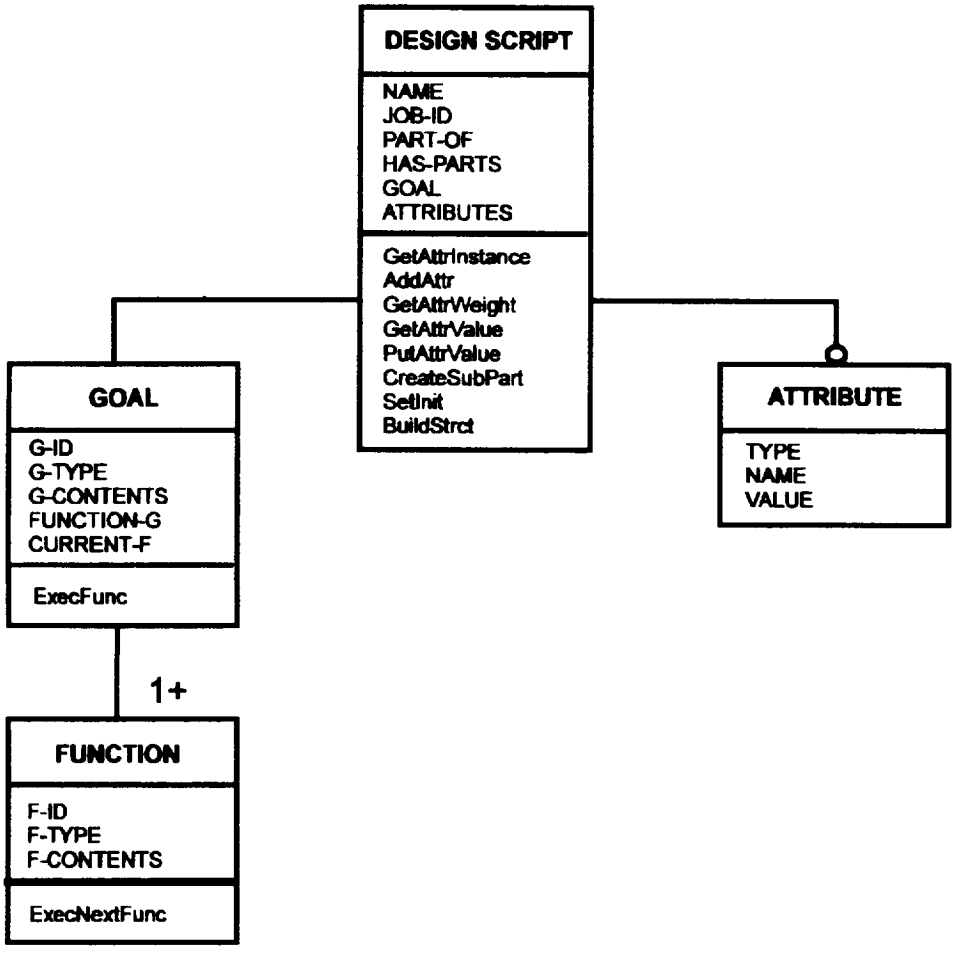

Fig. 2. Main Structures of Design Script

Usually, the initial goal of the design object is represented using natural language rather than by using a technical representation such as "power amp can drive as low as 2-ohm impedance loud speaker with 80 dB SPL." Then the goal is refined to the functions such as "require an amplifier

which can provide 20A current at peak with at least 200 W/ch power." The functions of the initial stage of the design process are regarded as goal of the next stage of design. The relations between goal and its functions are acquired knowledge from the designer's point of view. When a designer sets goals (or requirements) for the design object, the designer has what should be required to meet the goals. Although, the relations between goals and functions are intrinsic knowledge, they should be represented explicitly in the knowledge-based design process. The ***Goal*** slot of DS will hold a bound instance of the ***Goal-Design*** class as a slot value. Each instance of the ***Goal-Design*** class has its intrinsic functions kept as a multi-value slot to meet current goals. The ***Attributes*** slot of DS is a multi-value slot which keeps all the design variables and values for the design problem. DS provides various manipulation methods such as read-out and write-in the value of the attribute, decomposing the design into sub-parts, and creating instances of attributes and sub-parts. There always exist tradeoffs between generality and efficiency. To improve the efficiency of the design system, it could be built as a very domain specific. On the other hand, to make it general may require a sacrifice of efficiency. The DS has been developed to be very general, in that it can be applicable to different design domains.

GENERALIZATION AND AGGREGATION

Generalization is a powerful abstraction that can share commonalties among classes. Usually the generalization is represented as a superclass. The Design Scrip is a generalization of the design process that is based on the hierarchical structure of engineering objects. Each subclass inherits the features of its superclass. The inheritance mechanism of most object-oriented programming languages is based on a ***sub-type*** or ***is-a*** relation between classes. The COOL of CLIPS is not an exception. All the parts of a design object are represented as a subclass of DS so that they can inherit the properties of DS by using the built-in ***is-a*** hierarchy relation. But, the hierarchy relations among the parts themselves can not be represented by using ***is-a*** or ***kind-of*** relations, because they are actually ***part-of*** hierarchies (Fig. 3). Aggregation is the ***has-part*** or ***part-of*** relationship in which objects are part of the whole assembly. For example, let's take an example of a SO-8 package. We can say "*An SO-8 package is a kind of IC package.*" and "*IC is a kind of electronic component.*" ("SO-8" stands for "Small Outline 8 pin" and "IC" stands for "Integrated Circuit") We can see the ***is-a*** hierarchy relations between *an SO-8 package*, *IC package*, and *electronic component.* If we build the hierarchical structure for *SO-8*, we can see that it is composed of a mold material, a lead frame, wire bonds, etc. The lead frame is composed of lead1, lead2, die pad, etc. The relationship between these parts can not be represented with ***is-a*** relations. We can say the lead frame is not a type of SO-8, but a sub-part of the SO-8. So, in order to express the design knowledge, a mechanism must be available to handle the ***part-of*** relationship hierarchy. The slots ***PART-OF*** and ***HAS-PART*** of DS are used to define the ***part-of*** relations relative to super and subparts of the part under consideration.

DESIGN KNOWLEDGE MANAGEMENT

Many design engineers have stressed the uselessness of unmodifiable design knowledge. If fixed engineering design knowledge doesn't have a self-adaptive function for new situations, it may not work properly in these cases. For our system, the DS needs to follow the syntax of CLIPS when building a design knowledge base for a specific application design domain, because DS is

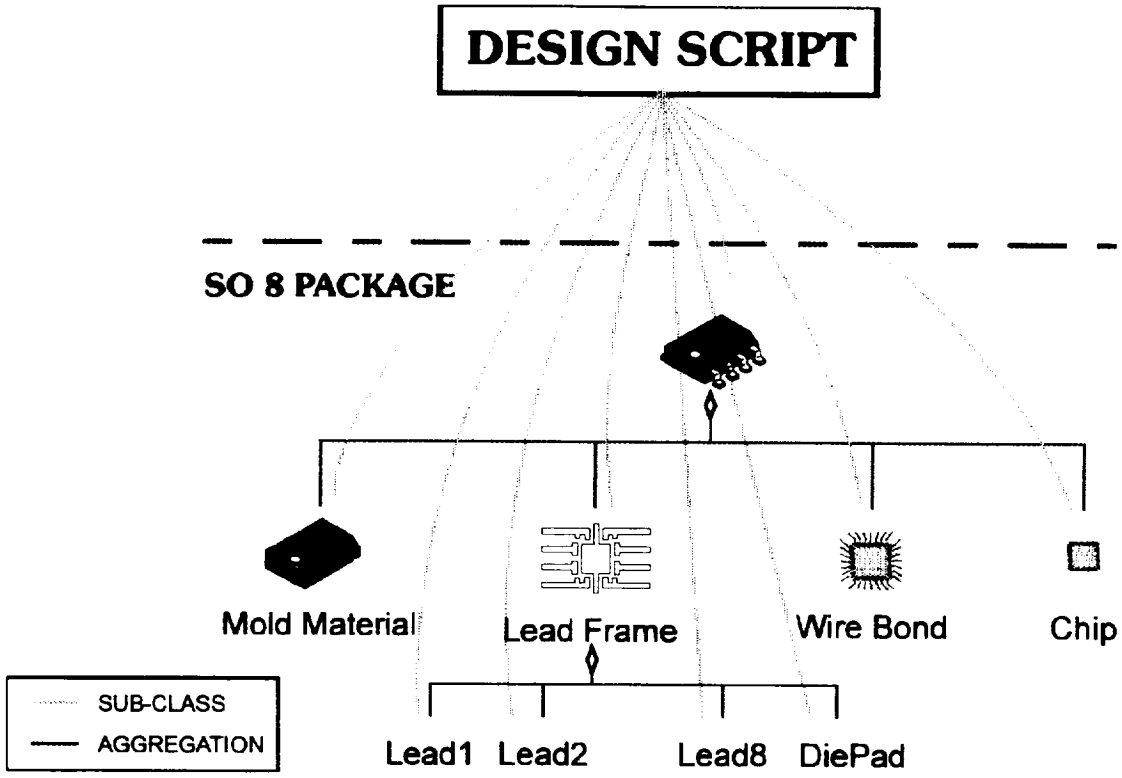

Fig. 3. Hierarchical Structure of SO8 IC Package and Design Script

implemented in CLIPS. The syntax of CLIPS is similar to that of the LISP language which uses lots of parentheses. CLIPS uses prefix notation in numerical computations rather than infix notation, which is generally used to represent algebraic expressions. If the developer is not familiar with this type of environment, it could be difficult to build the knowledge base. Considering that most design engineers are not computer scientists who can easily adapt to new programming environments, a user interface for knowledge entry and building is provided as a most essential part of the design system. The DS provides a GUI (Graphical User Interface) empty form as part of its functionality (Fig. 4). The design knowledge can be built easily by using the knowledge entry dialog-box on a part by part basis. What the designer has to do is fill in each field of the dialog box with the necessary information for designing or analyzing the object or subpart. It is an essential process to specify the design object in the hierarchical structure before entering design knowledge. Each dialog-box can edit design knowledge for a single (composite or elemental) part, such as its part-of and has-part relations, its goal, functions of the goal, design attributes for the part, and its numerical methods to get the implicit values of its design variables. When the user edits the dialog-box, the contents of the fields are stored temporarily in instances of the ***KENTRY*** class. Because the purpose of the ***KENTRY*** class is to provide an editable form of design knowledge that is easy to use, the contents of the design knowledge are kept as text in the corresponding slot. Once this information is entered by the user, it can be saved into a file in the form of instances of the ***KENTRY*** class from the main menu. Later, the user can modify the design knowledge by loading a previously created version

Knowledge Entry DialogBox

Object-Oriented Data in File

Fig. 4 Design System Structure with Knowledge Entry Dialog Box

from the file. Whenever the user wants to run the design system with the current design knowledge, he first compiles the knowledge from the instance form to the form of classes and their message-handlers which are in CLIPS syntax.

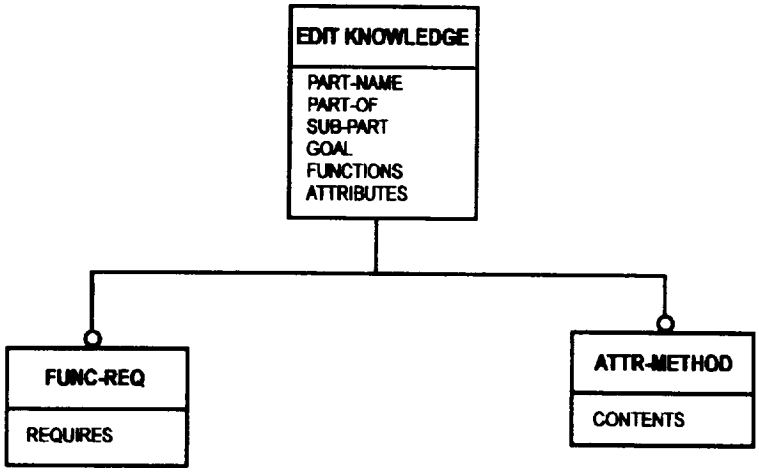

Fig. 5. Knowledge Entry Class

DESIGN APPLICATION: LEAD-FRAME DESIGN FOR PQFP

The DS has been applied in the domain of Lead-Frame design for a plastic quad flat pack IC package (PQFP). Acquiring design knowledge for the lead-frame of a PQFP is not a simple process. First of all, manufacturers tend to use their own knowledge to produce packages for their chips. Moreover, each company wants to keep this knowledge as proprietary information. There is available a limited amount of public design knowledge such as the standard package outlines contained in the *JEDEC Package Standard Outline*, research paper [11], or handbooks [19, 23]. This standard is widely used throughout the community of semiconductor manufacturers. The JEDEC standard is just for the outside dimensions, such as body size, length of outside lead, lead pitch, etc. The JEDEC manual doesn't refer to the inside dimensions of the package. This limitation reveals one difficulty of standardization in the packaging world. The major goal of the lead-frame design system for PQFP is to define the geometry of a lead-frame which can satisfy the JEDEC standard and the electrical and mechanical design constraints.

Fig. 6 describes the design knowledge for a lead-frame in graphical format. The values of lead-pitch, lead-width, and outside lead length are decided from JEDEC standard dimensions which are stored in a JEDEC database. The size of the chip bonding pad is decided by reference to the chip design technology base in use. The maximum wire span of the bonding wire from the bonding pad on the chip to the lead tip of the lead-frame is 100 mil. The lead tip pitch is around 10 mil. The width of the lead tip is around 6 mil. The length of the lead tip is dependent on its position. Each lead has an anchor hole. The fineness of the lead tip is also limited by the sheet metal blanking technology. The whole area of the metal part inside the plastic package is equal to or a slightly less than half of the entire plastic area.

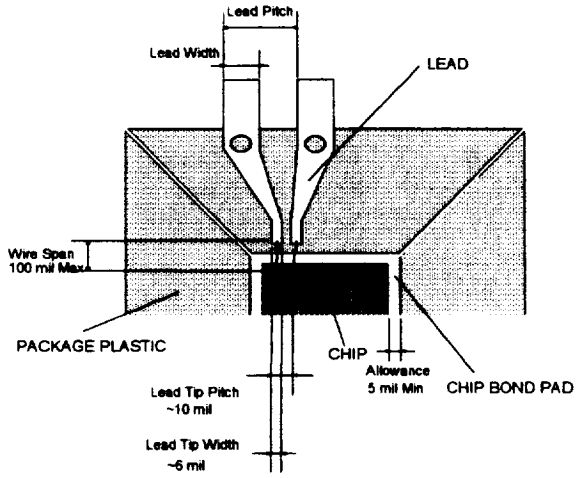

Fig. 6. Design Knowledge for the Lead-Frame of the PQFP.

When a user loads or edits the design knowledge using the *Knowledge Entry* dialog box and compiles it, the design knowledge is translated into CLIPS syntax (Fig. 7.). The first part of the knowledge is for the definition of LF-PQFP (Lead Frame of Plastic Quad Flat Pack) class which has *GOAL* and *HAS-PART* slots. The default value of the *GOAL* slot is the name of an instance of the GOAL-DESIGN class which has information about how to achieve the goal. Another multi-value slot contains the name and number of the subpart class. The sub-part of the lead-frame of the PQFP is composed of one die-bond-pad and the number of I/O leads in one octant of the package.

```
(defclass LF-PQFP (is-a DESIGN-SCRIPT)
 (role concrete)
 (slot GOAL (create-accessor read-write)(default "LF-1stGoal"))
 (slot WIRE-SPAN (create-accessor read-write)(default 100))
 (multislot HAS-PART (create-accessor read-write)(default "DieBondPad" "Lead1" "Lead2"
     "Lead3" . . . "Lead11"))
)

(message-handler LF-PQFP set-attributes()
 (bind ?self-i (instance-name ?self))
 (bind ?ins-name (symbol-to-instance-name (sym-cat ?self-i "LEADTIPPITCH")))
 (make-instance ?ins-name of AN-ATTRIBUTE
   (ATTR-NAME "LEAD-TIP-PITCH")
   (value 0.254))
 (send ?self add-attr ?ins-name)
 .

(message-handler LF-PQFP decide-Die-Size ()
 (bind ?SB (send ?self get-attr-value (send ?self get-attr-instance "Die-SB"))
 (bind ?Nio (send ?self get-attr-value (send ?self get-attr-instance "N-IO"))
 (bind ?Pd (send ?self get-attr-value (send ?self get-attr-instance "PAD-PITCH"))
 (bind ?DieSize (+ (* 2 ?SB) (*?Pd (+ 1 (/ ?Nio 4)))))
 (send ?self put-attr-value (send ?self get-attr-instance "PAD-PITCH") ?DieSize)
)
```

Fig. 7. CLIPS syntax for the Class and Message-Handler.

The reason for designing only an octant of I/O leads is that the geometry of the remaining leads can be derived from this portion of the geometry by considering the symmetry of the lead-frame. The next message-handler takes care of creating and storing the instances of the attribute. The last message-handler is an example of a method used to decide the design value of the attribute.

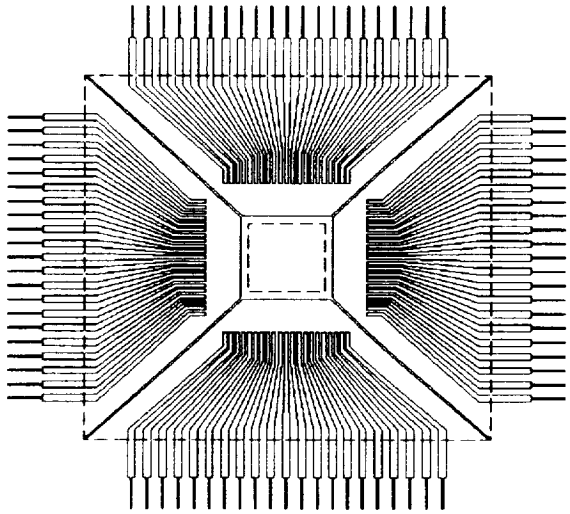

Fig. 8. Design Result: Lead-Frame of an 84 pin PQFP

Figure 8 shows the lead-frame of an 84 pin PQFP that was created by running the above lead-frame design system. The outer dotted square represents the plastic body of the PQFP and the inner dotted square shows the minimum size of the chip. The figure is drawn using the GNU plot program by providing the geometry that is produced by the lead-frame design system.

CONCLUSIONS AND FUTURE WORK

The main contribution of this research is in providing a general design-process control framework and a general design-knowledge representation scheme for physical objects to be designed using knowledge-based design systems. The Design Script developed here can be applied in any design domain because it contains domain knowledge independent about the design process. Usually, design is not only an individual activity but also requires cooperative team work that involves a number of designers from different fields. To support cooperative design work, DS provides an excellent framework that can be used in different domains. The capability of DS has been demonstrated in the domain of leadframe design for PQFPs.

This work is a part of the ongoing project. Much remains to be done to enhance the functionality of DS. Especially, management of design knowledge is a good candidate for future work. The current knowledge entry process doesn't provide any debugging functions. The design knowledge may be easily broken without careful knowledge entry. But there is also a need for a version management method like RCS (Revision Control System) or Aegis to enhance the productivity of cooperative design work. This enhancement is being considered to provide such functionality to the DS.

ACKNOWLEDGMENTS

The financial support from Semiconductor Research Corporation under the contract number MP-071 is gratefully appreciated.

REFERENCES

1. Akagi, S., "Expert System for Engineering Design Based on Object-Oriented Knowledge Representation Concept," *Artificial Intelligence in Design (ED. PHAM D. T.)*, 1991, 61-95.
2. Brown, D. C., and Chadrasekaran, B., "Design Problem Solving: Knowledge Structures and Control Strategies," *Pitman/Morgan Kaufmann*, 1989.
3. Chandrasekaran, B., "Generic Tasks in Knowledge-Based Reasoning: High-Level Building Blocks for Expert System Design," *IEEE Expert*, 1986, 1(3), 23-30.
4. Chandrasekaran, B., "Design Problem Solving: A Task analysis," *AI Magazine*, 1990, 11(4), 59 - 71.
5. Choi, C-K., and Kim, E-D., "A Preliminary Model of I-BUILDS: An Intelligent Building Design System," *Knowledge Based Expert Systems in Engineering: Planning and Design (Ed. Sriram D. and Adey R.A.)*, 1987, 331-343.
6. Coyne, R. D. et al., "Knowledge-Based Design Systems," *Addison-Wesley Publishing Co.*, 1990.
7. Dieter, G. E., "Engineering Design A Materials and Processing Approach," McGraw-Hill Book Co., 1983.
8. Gero, J. S., "Design Prototypes: A Knowledge Representation Schema for Design," *AI Magazine*, 1990, 11(4), 26-36.
9. Harty, N., "An Aid to Preliminary Design," *Knowledge Based Expert Systems in Engineering: Planning and Design (Ed. Sriram D. and Adey R. A.)*, 1987, 377-392.
10. Howe, A. E. et al., "Dominic: A Domain-Independent Program for Mechanical Engineering Design, *Artificial Intelligence in Engineering Design*, July 1986, 1(1), 23-28.
11. Jahsman, W. E., "Lead Frame and Wire Length Limitations to Bond Densification," *Journal of Electronic Packaging*, December 1989, Vol. 111, 289-293.
12. Maher, M. L., "HI-RISE and Beyond: Directions for Expert Systems in Design," *Computer Aided Design*, 1985, 17(9), 420-427.
13. Maher, M. L., "Process Models for Design Synthesis," *AI Magazine*, 1990, 11(4), 49-58.
14. Nguyen, G. T. and Rieu D., "SHOOD: A Design Object Model," *Artificial Intelligence in Design '92 (Ed. Gero J. S.)*, 1992, 221-240.
15. Oxman, R., "Design shells: a formalism for prototype refinement in knowledge-based design systems," *Artificial Intelligence in Engineering Design Progress in Engineering Vol.12*, 1993, 92-98.

16. Oxman, R., and Gero J. S., "Using an Expert System for Design Diagnosis and Design Synthesis," *Expert Systems*, 1987, 4(1), 4-15.

17. Rumbaugh, James et al., Object-Oriented Modeling and Design, Prentice Hall, 1991.

18. Runkel, J. T., et al., "Domain-Independent Design System, Environment for rapid development of configuration-design systems," *Artificial Intelligence in Design '92 (Ed. Gero J. S.)*, 1992, 21-40.

19. Seraphim, D. P.(Ed.), Principles of Electronic Packaging, *McGraw-Hill*, 1989.

20. STB, CLIPS Reference Manual Ver. 6.0, *Lyndon B. Johnson Space Center*, 1993.

21. Sriram, D. et al., "Knowledge-Based Expert Systems in Structural Design," *Computers and Structures*, 1985, 20(13) : 1-9.

22. Tham, K. W. and Gero, J. S., "PROBER -A Design System Based on Design Prototypes," *Artificial Intelligence in Design '92 (Ed. Gero J. S.)*, 1992, 657-675.

23. Tummala, R. R.(Ed.), Microelectronics Packaging Handbook, *Van Nostrand Reinhold*, 1989.